问题 A: 榨汁机

题目描述

zz 要做新鲜的橙汁。他有 n 个大小为 a1、a2、...、an 的橙子。zz 会按照固定的顺序将它们放入榨汁机，从 a1 号的橘子开始，然后是 a2 号的橘子，以此类推。要放入榨汁器，橘子的大小必须不超过 b，所以如果 zz 看到一个橘子太大，他会把它扔掉，然后继续下一个。榨汁机有一个专门的部分来收集废物。如果 zz 累积挤压的橙子的总尺寸大于 d，它就会溢出。当发生这种情况时，zz 会清空废料部分并继续挤压果汁。

问问 zz 要清空垃圾区多少次？

输入格式

输入的第一行包含三个整数 n、b 和 d（$1 \le n \le 100000, 1 \le b \le d \le 1000000$）– 橙子的数量、装在榨汁机中的橙子的最大尺寸和 d 值，d 值决定了应清空废料部分的条件。

第二行包含 n 个整数 a1，a2，...，an（$1 \le ai \le 1000000$）– 顺序放入榨汁机的橙子大小。

输出格式

输出一个整数即 zz 将不得不清空垃圾区的次数

Examples

Input

2 7 10

5 6

Output

1

Input

1 5 10

7

Output

0

Input

3 10 10

5 7 7

Output

1

Input

1 1 1

1

Output

0

Note

在第一个样品中，zz 将从两个橙子中挤出果汁，然后清空残渣部分。

在第二个例子中，橙子不适合榨汁机，所以 zz 根本没有果汁。

题解：

按题意直接模拟即可

代码如下：

```
#include<iostream>
#include<cstdio>
using namespace std;
int n,b,d;
int sum,ans;
int main()
{
	scanf("%d%d%d",&n,&b,&d);
	for(int i=1;i<=n;i++)
	{
		int a;
		scanf("%d",&a);
		if(a>b)continue;   //放不进榨汁机就跳过
		sum+=a;
		if(sum>d)     //如果榨汁机溢出  就清空
		{
			ans++;sum=0;     //ans 统计清空次数
		}
	}
	cout<<ans;
	return 0;
}
```

问题 B: 苹果人与面包人

题目描述

Appleman（苹果人）和 Toastman（面包人）玩游戏。最初，Appleman 给 Toastman 一组 n 个数字，然后他们开始完成以下任务：

每次 Toastman 得到一组数字时，他都会将所有数字相加，并将其加在分数上。然后他把这组数交给了 Appleman。

每次 Appleman 得到一个由单个数字组成的组时，他都会将这个组数舍弃。每次 Appleman 得到一组由多个数字组成的数时，他会将该组分成两个非空组（他可以用任何方式），并将每个组交给 Toastman。如此来回在 Appleman 和 Toastman 间传数，直到所有数被抛弃为止。

完成所有任务后，他们会查看得分值。他们能得到的最大得分值是多少？

输入格式

第一行包含单个整数 n（$1 \le n \le 3 \cdot 10^5$）。第二行包含 n 个整数 a1，a2。。。，an（$1 \le ai \le 10^6$）–给予 Toastman 的初始组。

输出格式

打印一个整数–可能的最大分数。

Examples

Input

3

3 1 5

Output

26

Input

1

10

Output

10

Note

在第一个示例中考虑以下情况。最初，Toastman 得到了小组[3,1,5]，并将分数加上 9，然后他将小组交给了 Appleman。Appleman 将组[3，1，5]分为两组：[3，5]和[1]。这两个都应该交给 Toastman。当 Toastman 收到分组[1]时，他将分数加 1，并将分组交给 Appleman（他会将其扔掉）。当 Toastman 收到分组[3,5]时，他将分数加 8，并将分组交给 Appleman。Appleman 以唯一可能的方式拆分[3]和[5]：[5]和[3]。然后他把两组都交给 Toastman。当 Toastman 收到[5]时，他在分数上加上 5，并将小组交给 Appleman（他会把它扔出去）。当 Toastman 收到[3]时，他在分数上加上 3 分，并将小组交给 Appleman（他会把它扔出去）。最后，Toastman 将分数增加了 9+1+8+5+3=26。这是最佳的操作顺序。

题解：本题大致题意就是一个人给出一些数，然后对这些数字进行相加，之后再把这些数字给人物二号，然后人物二号在对这些数字进行分组，但是这些数字的数量必须要大于 1，如果这些数字的数量等于 1，人物二号把这组数字扔掉，直到这些数字没有了就行了。

仔细理解样例的解释，本题显然是进行如何分组的策略选择问题，属于贪心问题。如果用对半分组，这些刚开始的时候就没有数消失，显然这种是不对的，因为就算刚开始的时候消失的少但是最后的时候就会一下少很多数字，就是相加的次数很少，不如我们每次分组的时候让最小的数字分成一组，其他的数字是一组，这样就可以很多组。

代码如下：

```
#include <iostream>
#include <algorithm>
using namespace std;
long long a[4000000];       //注意数据范围
int main()
{
    long long n,sum=0,ans=0;
    cin>>n;
    for(int i=1;i<=n;i++)
    {
        cin>>a[i];
        sum+=a[i];       //首先 Toastman 将所有数加起来
    }
    sort(a+1,a+n+1);     //排序  以便后续将最小的数划分出来
    int t=0;
    for(int i=1;i<=n;i++,t++)        //每次减去最小的数字
```

```
    {
        ans=ans+sum;        //ans 计算分值
        sum-=a[t];
    }
    cout<<ans<<endl;
}
```

```
#include<bits/stdc++.h>
using namespace std;
long a[300005];
int main()
{
    int n;
    long ans=0;
    cin>>n;
    for(int i=1;i<=n;i++)
    {
        cin>>a[i];
        ans+=a[i];
    }
    sort(a+1,a+n+1);
    for(int i=1;i<n;i++)
    {
        ans+=a[i]*i;
    }
    ans+=a[n]*(n-1);
    cout<<ans<<endl;
    return 0;
}
```

问题 C: 编程课的密码

题目描述

达莎爬完楼梯后就开始上课了。她需要写一个密码才能开始上课。密码是长度为 n 的字符串，满足以下要求：

字符串中至少有一个数字，

字符串中至少有一个拉丁字母的小写（小）字母，

字符串中至少有三个列出的符号之一：“#”、“*”、“&”。

考虑到这是编程课的密码，编写密码并不容易。

对于密码的每个字符，分别从一个字符矩阵中获取。字符矩阵的每一行是一个长度为 m 的字符串，分别从每一行中获取一个字符组成最后的密码。初始时，n 行字符串中的每个字符串上都有一个指第一个字符的指针，即所有指针都位于相应字符串中索引为 1 的字符上（所有位置从 1 开始编号）。

在一次操作中，达莎可以将一个字符串中的指针向左或向右移动一个字符。字符串是循环的，这意味着当我们将索引为 1 的字符上的指针向左移动时，它会移动到索引为 m 的字符，当我们将指针从位置 m 向右移动时，指针会移动到位置 1。

您需要确定使屏幕上显示的字符串成为有效密码所需的最小操作次数。

输入格式

第一行包含两个整数 n，m（$3 \le n \le 50, 1 \le m \le 50$）分别表示-密码长度和分配给密码符号的字符串长度。

接下来的 n 行中的每一行都包含分配给密码字符串的第 i 个符号的字符串。其长度为 m，由数字、小写英文字母和字符“#”、“*”或“&”组成。

数据保证总能得到一个有效的密码。

输出格式

输出一个整数表示-使屏幕上显示的字符串成为有效密码所需的最小操作次数。

Input

3 4

1**2

a3*0

c4**

Output

1

Input

5 5

#*&#*

*a1c&

&q2w*

#a3c#

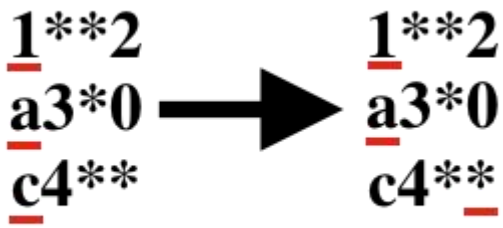
&#&

Output

3

注：

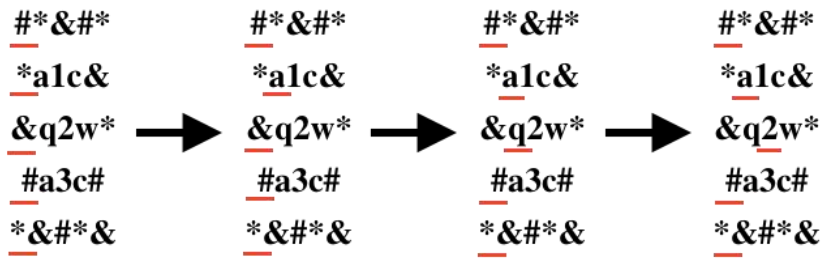
在第一次测试中，需要将第三个字符串的指针向左移动，以获得最佳答案。



在第二个测试中，可能的算法之一是：

1.将第二个符号的指针向右移动一次。

2.将第三个符号的指针向右移动两次。



题解：

分别枚举提供数字、字母、其它符号是哪个字符串提供的，然后分别计算每个的步数，求和，最后取最小值即可。

注意的是因为是一个环，所以计算步数的时候会有不同。

```
#include<bits/stdc++.h>
using namespace std;
int n,m,ans=1e9;
string s[60];
int f(int l,int r)
{
    return min(r-l,m-r+l);
}//计算步数
int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)
        cin>>s[i];
    for(int i=1;i<=n;i++)//枚举数字
    {
        int x=1e9;
        for(int j=0;j<m;j++)
            if(s[i][j]>='0'&&s[i][j]<='9')
                x=min(x,f(0,j));
        for(int j=1;j<=n;j++)//枚举字母
        {
            if(i==j)continue;
            int y=1e9;
            for(int k=0;k<m;k++)
                if(s[j][k]>='a'&&s[j][k]<='z')
                    y=min(y,f(0,k));
            for(int k=1;k<=n;k++)//枚举其它符号
```

```
            {
                int z=1e9;
                if(i==k||j==k)continue;
                for(int l=0;l<m;l++)
                    if(s[k][l]=='#'||s[k][l]=='*'||s[k][l]=='&')
                        z=min(z,f(0,l)),ans=min(ans,x+y+z);//更新答案
            }
        }
    }
    cout<<ans;
    return 0;
}
```

也可以用动态规划求解：

令 dp[i][j] 为第 i 行选择 j 类字符的最小移动次数

（j == 1 数字）

（j == 2 小写字母）

（j == 3 其他符号）

把每一行单独处理，最后枚举统计最小的答案。

答案为 min(dp[i][1] + dp[j][2] + dp[k][3])

代码如下：

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int INF = 1e9 + 7;
const int SZ = 100 + 5;
int n,m;//1 (0 - 9) 2 (a - z) 3 (# & *)
char s[SZ][SZ];    //dp[i][j] 第 i 行换成 j 类型的最小移动次数
ll dp[SZ][4],ans;
int main()
{
    scanf("%d%d",&n,&m);
    for(int i = 1;i <= n;i ++ )
        for(int j = 1;j <= 3;j ++ )
        dp[i][j] = INF;     //初始化
    for(int i = 1;i <= n;i ++ ) scanf("%s",s[i] + 1);
    for(int i = 1;i <= n;i ++)
    {
        for(int j = 1;j <= 3;j ++ )
        {
```

```
            for(int k = 1;k <= m;k ++ )
            {
                if(s[i][k] >= '0' && s[i][k] <= '9') dp[i][1] =
min(dp[i][1],min((ll)(k - 1),(ll)(m - k + 1)));       //判断是数字的情况
                if(s[i][k] >= 'a' && s[i][k] <= 'z') dp[i][2] =
min(dp[i][2],min((ll)(k - 1),(ll)(m - k + 1)));       //判断是小写字母的情
况
                if(s[i][k] == '#' || s[i][k] == '*' || s[i][k] == '&' )
dp[i][3] = min(dp[i][3],min((ll)(k - 1),(ll)(m - k + 1)));      //判断是
符号的情况
            }
        }
    }
    ans = INF;      //初始化
    for(int i = 1;i <= n;i ++ )
        for(int j = 1;j <= n;j ++ )
            for(int k = 1;k <= n;k ++ )
            {
                if(i != j && i != k && j != k)
                {
                    ans = min(ans,dp[i][1] + dp[j][2] + dp[k][3]);
//将三种加起来找最小
                }
            }
    printf("%d",ans);
    return 0;
```

问题 D: 三分线

题目描述

zz 关注一场篮球比赛，并标记每队投掷的距离。他知道每次成功的投掷都有 2 分或 3 分的价值。如果投掷的距离不超过 d 米，则投掷的分数为 2 分；如果距离大于 d 米，则投掷的分数为 3 分，其中 d 为某个非负整数。

zz 希望第一队的得分优势（第一队的得分减去第二队的得分）是最大的。为此，他想选择一个最最优的 d 来实现这个目标。

输入格式

第一行包含整数 n（$1 \le n \le 2 \cdot 10^5$）是第一队的投掷次数。然后是 n 个整数，第一个队伍投掷的距离 a_i（$1 \le a_i \le 2 \cdot 10^9$）。

然后是数字 m（$1 \le m \le 2 \cdot 10^5$）是第二队的投掷数量。然后是 m 个整数，第二支队伍的投掷距离 b_i（$1 \le b_i \le 2 \cdot 10^9$）。

输出格式

输出两个格式为 a:b 的数字。

当有多个相同的最大差值的情况时，输出第一个人分数最大的那组。

Examples

Input

3

1 2 3

2

5 6

Output

9:6

Input

5

6 7 8 9 10

5

1 2 3 4 5

Output

15:10

题解：

枚举三分线(离散后)的位置

然后根据预处理的前缀和,快速算出两个队伍的分数。

代码如下：

```
#include <bits/stdc++.h>
using namespace std;
const int N = 2e5;

int n, m;
int a[N + 10], b[N + 10], aa[2 * N + 100], bb[2 * N + 100];    //注意数据范围
map<int, int> dic;
map<int, int> dic2;

int main()
{
```

```
cin >> n;
for (int i = 1; i <= n; i++)
{
	cin >> a[i];
	dic[a[i]] = 1;	//用桶来记录 a 组每个距离的个数
}
cin >> m;
for (int i = 1; i <= m; i++)
{
	cin >> b[i];
	dic[b[i]] = 1;	//用桶来记录 b 组每个距离的个数
}
map<int, int>::iterator it;
int cnt = 0;
dic2[0] = ++cnt;
for (it = dic.begin(); it != dic.end(); it++)	//利用离散枚举
{

	dic2[(*it).first] = ++cnt;
}
for (int i = 1; i <= n; i++)
{
	aa[dic2[a[i]]]++;
}

for (int i = 1; i <= m; i++)
{
	bb[dic2[b[i]]]++;
}
for (int i = 1; i <= cnt; i++)
	aa[i] += aa[i - 1];	//利用前缀和 aa[i]到当前为止的 2 分数量
for (int i = 1; i <= cnt; i++)
	bb[i] += bb[i - 1];	//利用前缀和 bb[i]到当前为止的 2 分数量
int ans = -(1e8), maxa;
for (int i = 1; i <= cnt; i++)
{
	int ga = aa[i] * 2 + (n - aa[i]) * 3;
	int gb = bb[i] * 2 + (m - bb[i]) * 3;
	if (ga - gb > ans)
	{
		ans = ga - gb;
		maxa = ga;
	}	//找到最大差值
	else if (ga - gb == ans)
```

```
            {
                if (maxa < ga)
                {
                    maxa = ga;      //差值相同，寻找分值最大
                }
            }
        }
        cout << maxa << ":" << maxa - ans << endl;

        return 0;
}
```

问题 E: 苏打水

题目描述

zz 生日过后还剩下 n 瓶苏打水。每瓶苏打水可以用两个值来描述：苏打水的剩余量 ai 和瓶体积 bi（ai≤bi）。

zz 决定将所有剩余的苏打水倒入最少数量的瓶子中，而且他必须尽快完成。zz 花了 x 秒将 x 单位的苏打水从一个瓶子倒到另一个瓶子。

zz 请你帮他确定 整数 k 和整数 t:

k - 存放所有剩余苏打水的最少瓶子数量

t - 将苏打水倒入 k 个瓶子的最短时间。

注意一个瓶子不能储存比它的体积更多的苏打水。应保存所有剩余的苏打水。

输入格式

第一行包含正整数 n (1≤n≤100) - 瓶子的数量。

第二行包含 n 个正整数 a1,a2,...,an (1≤ai≤100)，其中 ai 是第 i 个瓶子中剩余的苏打水量。

第三行包含 n 个正整数 b1,b2,...,bn (1≤bi≤100)，其中 bi 是第 i 个瓶子的体积。

保证任意 i 的 ai≤bi。

输出格式

唯一的一行应该包含两个整数 k 和 t，其中 k 是可以存储所有苏打水的最少瓶子数，t 是将苏打水倒入 k 个瓶子的最短时间。

Input

4

3 3 4 3

4 7 6 5

Output

2 6

Input

2

1 1

100 100

Output

1 1

Input

5

10 30 5 6 24

10 41 7 8 24

Output

3 11

注意

在第一个例子中，尼克可以将苏打水从第一瓶倒到第二瓶。这将需要 3 秒钟。之后，第二瓶将包含 3+3=6 单位的苏打水。然后他可以从第四瓶倒到第二瓶和第三瓶：一个单位倒到第二个瓶子，两个单位倒到第三个瓶子。这将需要 1+2=3 秒。所以，所有的汽水都会装在两个瓶子里，他会花 3+3=6 秒来做这件事。

题解：dp[k][j]表示的是选择 k 个瓶子，瓶子里本来有 j 的苏打水，最大可以达到这么多瓶子的总容积

代码如下：

```
#include <iostream>
#include <cstdio>
#include <cmath>
#include <cstring>
#include <algorithm>
#define maxn 106
using namespace std;
struct Soda
{
    int res;
    int v;
```

```
}soda[maxn];     //res 余量  v 体积
bool cmp(const Soda & a,const Soda & b)
{
      return a.v>b.v;       //按照体积大小  从小到大排序
}
int n,m;
int sodaamount;
int dp[105][10010];
int main()
{
      while(scanf("%d",&n)==1){
      sodaamount=0;
      int a;
      for(int i=1;i<=n;i++)
      {
            scanf("%d",&soda[i].res);
            sodaamount+=soda[i].res;       //统计剩下所有余量
      }
      for(int i=1;i<=n;i++)
      {
            scanf("%d",&soda[i].v);
      }
      sort(soda+1,soda+n+1,cmp);       //按照体积大小排序
      int cnt=0;
      int    t=0;
      for(cnt=1;cnt<=n;cnt++)
      {
            t+=soda[cnt].v;
            if(t>=sodaamount)break;
            //当前瓶子的所有剩余体积可以装下所有余量时，停止遍历，寻找最小
瓶子数量
      }
      memset(dp,-1,sizeof(dp));     //dp 数组初始化
      int ans=0;     //ans 记录最短时间
      dp[0][0]=0;         //初始化
      for(int i=1;i<=n;i++)
      {
            for(int j=sodaamount;j>=soda[i].res;j--)       //余量
            {
                  for(int k=i;k>=1;k--)
                  {
                        if(dp[k-1][j-soda[i].res]!=-1)       //
                        dp[k][j]=max(dp[k][j],dp[k-1][j-soda[i].res]+soda[i].v);
                        //第 k-1 个瓶子原来的量是 j-soda[i].res  状态转移
```

```
                }
            }
        }

        for(int i=sodaamount;i>=0;i--)
        {
            if(dp[cnt][i]>=sodaamount)        //瓶子容量可以装下，停止循环
            {
                ans=sodaamount-i;             //记录当前所需的时间
                break;
            }
        }
        printf("%d %d\n",cnt,ans);
        }
        return 0;
}
```